AMENDMENTS TO THE SPECIFICATION

Please amend paragraph 0005 as shown.

[0004] The RDMA protocol is used to make a section of main system memory on a first machine directly available to a remote second machine. The protocol associates the memory in the first machine with a handle referred to as a STag. To offload as much processing as possible from the CPU in the first machine, a NIC in the first machine utilizes the STag generated by a consumer comprising, for example, an application program. The STag is then sent to the second machine, which can perform a write by sending the STag back with associated data. Upon receiving this data and associated STag, the NIC in the first machine will read the STag and use a DMA transfer to move the data into the memory associated with the STag.

Please amend paragraph 0005 as shown.

[0005] Data traveling over the internet can take several different routes from one machine to another. The path through the Internet will change when loading on servers changes or when servers fail all together. This can cause difficulty for a machine with multiple NICs when performing an RDMA transfer. As the route the data takes through the Internet changes, it is possible that the path chosen from the first machine to the second machine will change in a manner that causes the path between these two machines to change from NIC 1 to NIC 2 in machine 1. Recall that the NIC 1 ~~sends~~ generates the STag. Therefore, NIC 2 will have no knowledge of a STag generated by NIC 1. If the route from machine 1 to machine 2 uses NIC 1 when an STag in ~~sent~~ generated, and then the route changes to one which uses NIC 2 before machine 2 sends data to machine 1, machine 2 will return data with an STag that is unknown to NIC 2.

Please amend paragraph 0006 as shown.

[0006] There is a need for a method to handle STag's ~~sent~~ generated by one NIC and received by another NIC in the same machine.

Please amend paragraph 0007 as shown.

[0007] Embodiments are directed to methods that overcome the problem of a STag arriving at a network interface that did not ~~send~~ generate the STag. The method relies on network interfaces

on a given computer having unique STags. This can be assured by the operating system. Because STags on a given computer are unique, a network interface receiving an STag ~~sent~~ generated by another network interface on the same computer is enabled to detect that the STag was ~~sent~~ generated by a different network interface. When such a STag is detected, the network interface receiving the STag passes this STag to a higher level of software, which can be an RDMA program component which resides in the OS kernel. The RDMA program component can identify which NETWORK INTERFACE ~~sent~~ generated the STag and query the associated network interface for all STags ~~utilized~~ generated by this network interface and the associated addresses of the allocated memory. The address is then passed to the network interface that received the unknown STag. With the memory address, the network interface can then complete the data transfer.

Please amend paragraph 0008 as shown.

[0008] More specifically, an embodiment is directed to a method for transferring control between a first network interface and at least a second network interface in a multiple network interface device after the first network interface transmits an identifier generated for use by the first network interface to a second device. The identifier can be associated with a memory location in the multiple network interface device, and the identifier and an associated data field are capable of being received by the second network interface. The method further includes receiving a message from the second network interface to a program component, the message indicating the reception of the identifier from the second device. Next, the method provides for querying the first network interface to supply the program component with a list of identifiers ~~sent~~ generated by the first network interface and associated memory locations in the multiple network interface device memory. If the identifier received by the second device is present in the list, the method provides for transmitting a memory location associated with the identifier to the second network interface. Thus, the second network interface becomes capable of transmitting the associated data field to the memory location associated with the identifier.

Please amend paragraph 0009 as shown.

[0009] Another embodiment is directed to a method for transferring control between a first network interface and at least a second network interface in a host computer including the first

network interface and the second network interface. The method includes receiving an identifier from a remote computer, the identifier ~~sent~~ generated by the first network interface and associated with a memory location in the host computer. Next, the method provides for sending a message to a program component indicating the reception of the identifier, the program component configured to query the first network interface for a list of identifiers ~~sent~~ generated by the first network interface and associated memory locations in the host computer. If the list of identifiers includes the identifier from the remote computer, the method provides for receiving a memory location associated with the identifier. If the list of identifiers does not include the identifier from the remote computer, the method provides for invalidating the identifier from the remote computer.

Please amend paragraph 0024 as shown.

[0024] Device 100 may also have one or more input devices 114 such as keyboard ~~162~~, mouse ~~161~~, pen, voice input device, touch-input device, etc. One or more output devices such as a display ~~191~~, speakers ~~197~~, printer ~~196~~, etc. may also be included. All these devices are well known in the art and need not be discussed at greater length here.

Please amend paragraph 0033 as shown.

[0033] Referring now to FIG. 3 and FIG. 4 in combination, the flow diagram of FIG. 4 illustrates an RDMA read operation. The transfer is initiated by an application 303 making a request of OS 305 to transfer data in block 402. All operating system commands in this case can occur in kernel mode. The OS 305, which may include network layer 307, determines that the request requires a network access. If NIC 309 is capable of RDMA, OS 305 can make this transfer utilizing RDMA to offload some of the processing burden from the host CPU. The RDMA transfer starts by OS 305 requesting a STag from NIC 309 in block 404. NIC 309 will return an identifier referred to as STag 401 to network OS 305 in block 406. OS 305 will then allocate memory for the transfer creating memory allocation 311 and send the address of the allocated memory 311 to NIC 309 in block 408. NIC 309 then associates memory allocation 311 with STag 401. This association allows NIC 309 to place any data arriving with STag 401 in memory allocation 311. NIC 309 then creates RDMA packet 403 in block 412. RDMA packet 403 consists of STag 401 and a command to read data. NIC 309 next encapsulates RDMA packet

403 in a TCP/IP datagram 405 in block 414. In block 416, TCP/IP datagram 405 is transmitted onto the Internet 313 which routes the TCP/IP datagram 405 to remote computer 315 in block 416. Remote computer 315 extracts RDMA packet 403 from TCP/IP datagram 405 in block 418. This extraction can be done in a NIC or in software. In block 420, STag 401 is extracted from RDMA packet 403 and in combination with the requested data 407 is used to form RDMA packet 411. In block 422 TCP/IP datagram 409 encapsulating RDMA packet 411 TCP/IP datagram 409 is then sent onto Internet 313 which routes TCP/IP datagram 409 to computer 301 in block 424. NIC 309 then receives TCP/IP datagram 409 in block 426. NIC 309 then extracts RDMA packet 411 from TCP/IP datagram 409 in block 428. In block 430 NIC 309 extracts STag 401 from RDMA packet 411 and checks the invalidation bit in the RDMA header. If this bit is set, the STag is invalidated. STag 401 is then used to retrieve the associated memory allocation 311 and the requested data 407 is sent to this memory allocation using DMA in block 432.

Please amend paragraph 0037 as shown.

[0037] FIG. 6a illustrates that an application 503 that may be implemented in remote computer ~~503~~ 501 may request transfer of data from operating system OS 505, in step 602. In step 604, OS 505 may request STag from NIC 509, in response to which NIC 509 returns STtag 601 to OS 505, in step 605. OS 505 then may allocate memory location 511 and notify NIC 509 about it, in step 608, upon which NIC 509 may associate memory location 511 with STag 601, in step 610. In step 612, NIC 509 forms RDMA packet 603 with the read request. In step 614, NIC 509 forms TCP/IP datagram 611 encapsulating RDMA packet 603. The TCP/IP datagram 611 may then be sent by NIC 509 to remote computer 515 over the Internet 513, in step 616. Upon receiving the TCP/IP datagram 611, remote computer 515 may extract RDMA packet 603 from TCP/IP datagram 611, in step 618. In step 620, remote computer 515 forms an RDMA packet 606 with STag 601 and requested data 607. In step 622, remote computer 515 may form TCP/IP datagram 609 encapsulating RDMA packet 606. In step 624, remote computer 515 sends TCP/IP datagram 609 to computer 501 over the Internet 513. As was discussed above, the route used by the Internet may change so that, in step 626, second NIC (e.g., NIC 510) may receive TCP/IP datagram from second device.

Please amend paragraph 0038 as shown.

[0038] When NIC 510 receives TCP/IP datagram 609 in block 626 of FIG. 6b, NIC 510 will perform all necessary TCP/IP processing and then extracts RDMA packet 606 in block 628. Next, STag 601 is removed from RDMA packet 606 in block 630. In block 632, NIC 510 will search a list of all valid STags ~~utilized by~~ created in NIC 510 but will fail to find STag 601. There are two possibilities at this point, either STag 601 is invalid or STag 601 was created by another NIC on the same computer 501. NIC 510 assumes that the later is true and, in block 634, reports STag 601 to OS 505 as an unrecognized STag. OS 505 attempts to ascertain if STag 601 is valid. To accomplish validation, in block 636 OS 505 queries NIC 509 for all valid STags and associated addresses ~~sent~~ generated by NIC 509. In block 638 NIC 509 returns the requested list. OS 505 then searches this list for STag 601, in block 640. In block 642, OS 505 makes a decision as to whether STag 601 is found. If not, block 650 is implemented in which STag 601 is reported as invalid to NIC 510. NIC 510 will then discard the packet because there is no valid memory location to transfer the packet. If a valid memory location were available, transferring a packet with an invalid STag into memory would present a security risk. Note that while in this example this branch is clearly not possible as we stated that NIC 509 ~~sent~~ generated STag 601. It is included however to illustrate the complete algorithm which must correctly handle invalid STags. If block 642 determines that STag 601 is in the list of STags ~~sent~~ generated by NIC 509, then, in block 644, OS 505 finds memory location 511 associated with STag 601. In block 646 associated memory location 511 is reported to NIC 510. The knowledge of associated memory allocation 511 allows NIC 510 to transfer requested data 607 to memory allocation 511 completing the transfer in block 648.